Probabilistic Models
for density estimation

Machine Learning 2020
mlvu.github.io

probabilistic models

part 1:
Maximum Likelihood

Normal Distributions (aka Gaussians)
univariate, regression, multivariate, mixture-of-gaussians

part 2:
Alternating optimization
k-means clustering

Expectation-Maximization:
- Intuition
- Solutions, Proof of convergence

a simple example

p(Heads | Straight) =1/2 p(Heads | Bent) = 4/5

p(Tails | Straight) = 1/2 p(Tails | Bent) = 1/5

HTH HHT HHT HTH

Let's start with a simple motivating example. We have two coins, a bent one and a straight one. Flipping these coins gives us different probabilities of heads and tails.

We ask a friend to pick a random coin without showing us, and to flip it twelve times. The resulting sequence has more heads than tails, but not such a disparity that you would never expect it from a fair coin. If we had to guess which coin our friend had picked, which should we guess?

image source: **https://www.magictricks.com/bent.html**

a simple example

p(Heads | Straight) =1/2 p(Heads | Bent) = 4/5

p(Tails | Straight) = 1/2 p(Tails | Bent) = 1/5

Observed data

This is a simple version of a model selection problem. We've observed some data, and we want to select which single model is most suitable given the observed data.

Note that picking just one model is a frequentist approach, a Bayesian would simply compute a probability distribution over the two points.

image source: **https://www.magictricks.com/bent.html**

maximum likelihood

$$\underset{\text{Coin} \in \{\text{Bent,Straight}\}}{\arg\max} \; p(\text{HTHHHTHHTHTH} \mid \text{Coin})$$

$$\underset{\text{Model} \in \text{Model Space}}{\arg\max} \; p(\text{Data} \mid \text{Model})$$

which coin?

HTHHHTHHTHTH

$$p(\text{D}|\text{Bent}) = \frac{4}{5}\frac{1}{5}\frac{4}{5}\frac{4}{5}\frac{4}{5}\frac{1}{5}\frac{4}{5}\frac{4}{5}\frac{1}{5}\frac{4}{5}\frac{1}{5}\frac{4}{5} \approx 0.000268$$

$$p(\text{D}|\text{Straight}) = \frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2} \approx 0.000244$$

Since the coin flips are independent, the probability over the whole sequence is just the product over the probabilities of the individual flips. There's not much in it, but the likelihood for the bent coin is slightly higher, so that's the preferred model under the maximum likelihood criterion.

(LOG) LIKELIHOOD: What we *maximise* to fit a probability model

LOSS: What we *minimise* to fit a machine learning model

We often take the logarithm of the likelihood. The logarithm is a monotonic function so the likelihood and the log likelihood have their minima in the same place, but the log likelihood is often easier to manipulate symbolically (see the first homework exercise).

The log likelihood of a probability distribution is a lot like the loss functions we've already encountered many times.

In fact, if we want to fit a probability distribution inside a deep learning system, we usually take the negative log likelihood, so that we can do gradient descent.

models

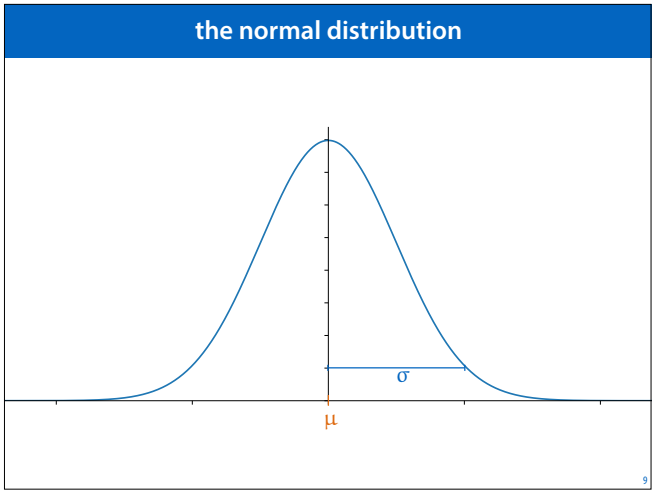
1 dimensional normal distribution
aka a univariate Gaussian

regression with Gaussian errors
squared error regression assumes normally distributed residuals

n-dimensional normal distribution
aka a multivariate Gaussian

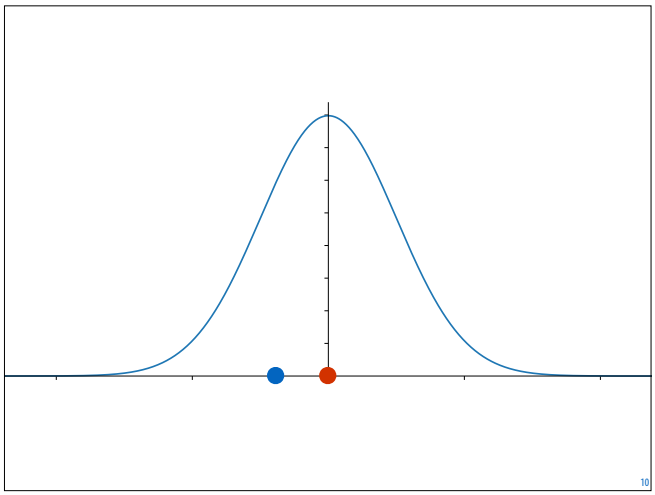
mixture of Gaussians

We'll look four examples of maximum likelihood fitting in practice.



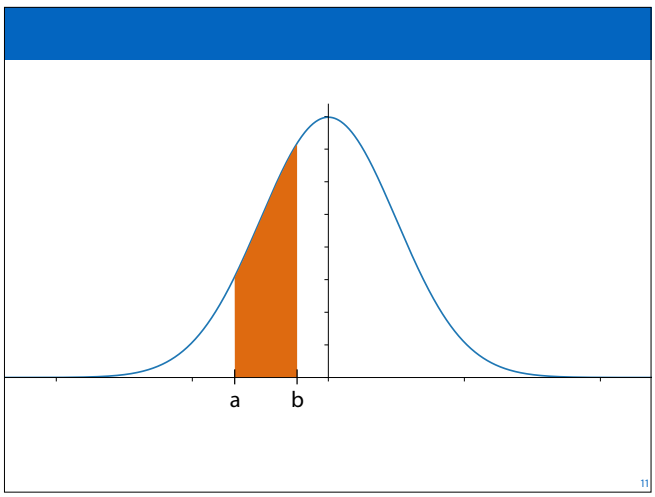
Here is the one dimensional normal distribution.

One of the reasons that the normal distribution is so popular is that it has a definite *scale*. If I look at something like income distribution, the possible values cover many orders of magnitude, from 0 to billions. This is not the case with normally distributed phenomena. Take height for instance: no matter how many people I check, I will never see a person that is 5 meters tall.

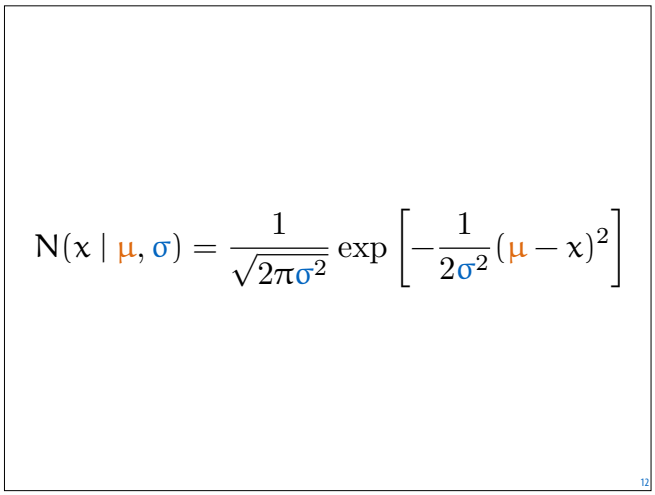


Let's say we sample a point from this probability distribution. Which point has the highest probability?

This is a trick question, because **both points have probability 0**. In these kinds of distributions over a continuous sample space, no single point has a probability. The blue curve defines a probability density.

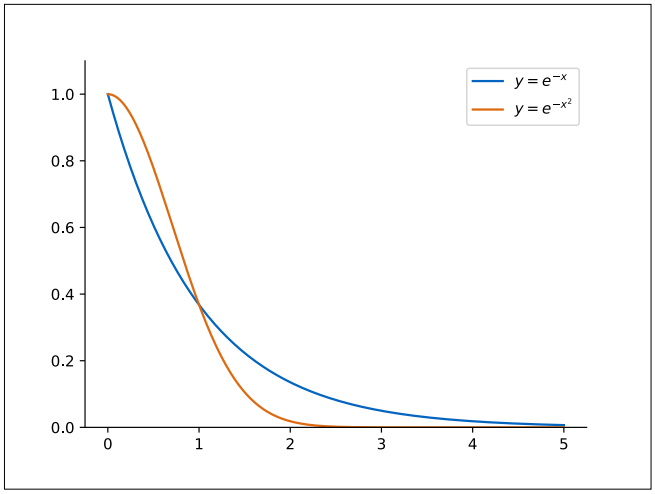


Only **intervals** have a probability: the probability that we sample a point between a and b is equal to the surface area of the orange shape (the area under the curve between a and b).

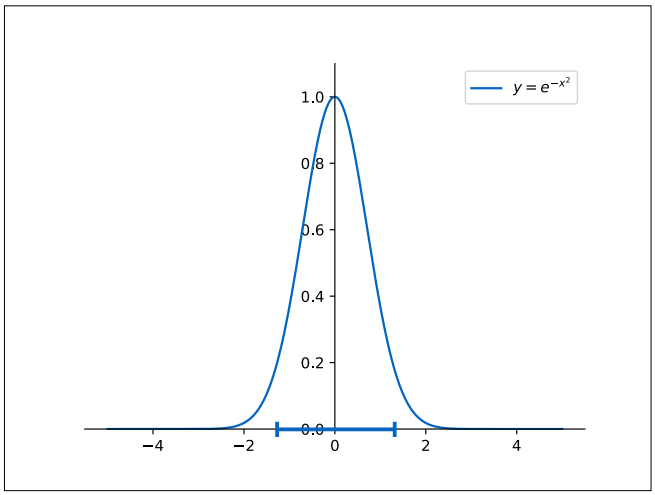


This is the formula for the probability density function of the univariate normal distribution.

It looks very imposing, but if you know how to interpret it, it's actually not that complicated. Let's go through it step by step, before we try to fit it to some data.

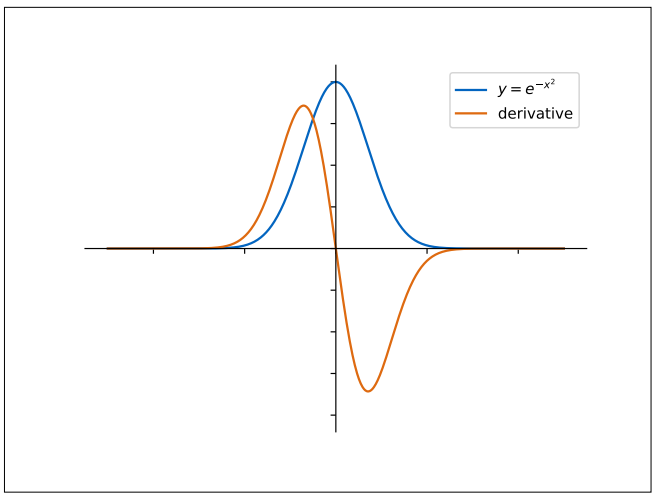


The first thing we want, to ensure that our distribution has a definite scale (i.e. outliers are incredibly unlikely), is an exponentially decaying tail. e^{-x} gives us such a decay. However, e^{-x^2} has an even stronger decay, and it has two more benefits: The function flattens out at the peak, giving us a nice bell-shaped curve, and it has an inflection point: the point (around 1.7) where the curve moves from decaying with increasing speed to decaying with decreasing speed.

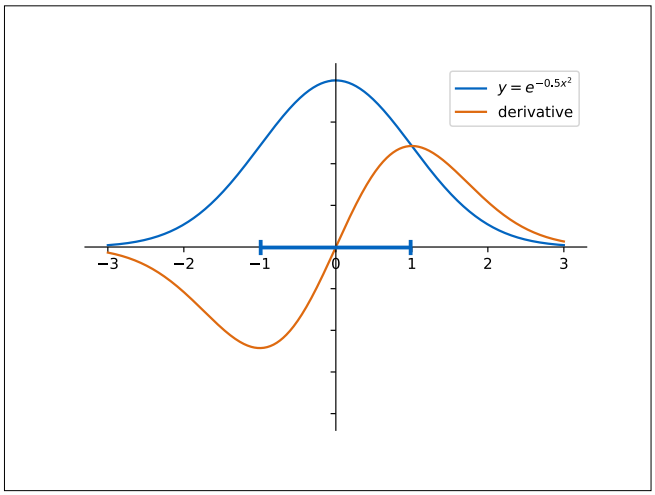


Here is what it looks like across the whole real number range. The two inflection points are natural choices for the range bounding the "characteristic" scale of this distribution. The range of outcomes which we can reasonably expect.

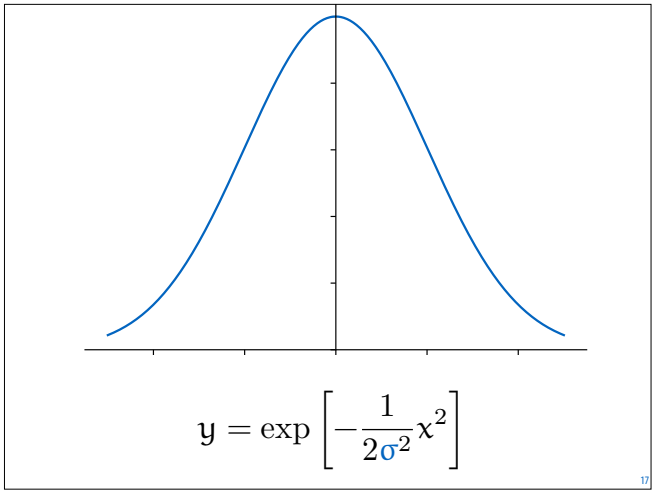
This is a little subjective: any outcome is possible, and the characteristic scale depends on what we're willing to call unlikely. But given the subjectivity, the inflection points are as good a choice as anything.



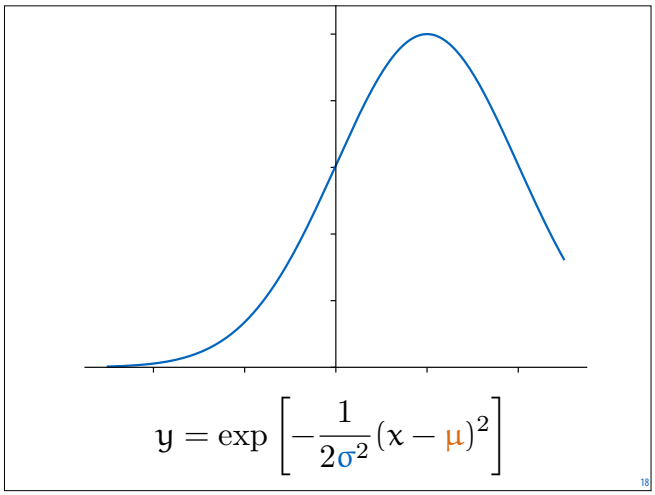
The inflection points are the peaks of the derivative (i.e. where the second derivative crosses the horizontal axis).



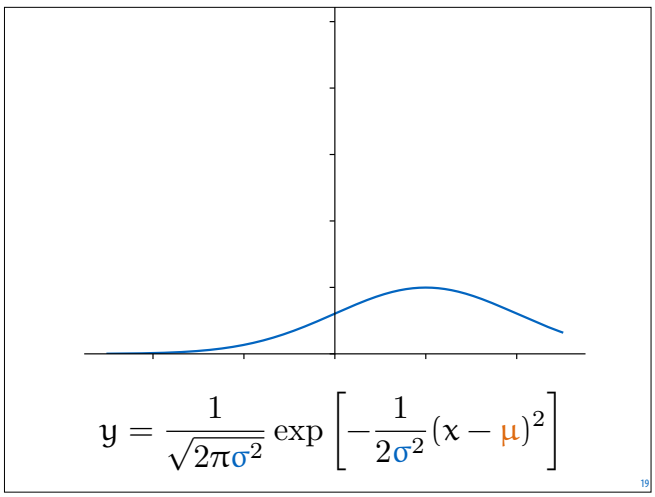
If we add a 0.5 multiplier to the inputs, the inflection points hit -1 and 1 exactly. This gives us a curve for which the characteristic scale is [-1, 1], which seems like a useful starting point (we can rescale this later to any range we require).



To change the scale, we add our first parameter, the variance. If we set the variance to 2, the characteristic scale changes to [-4, 4]

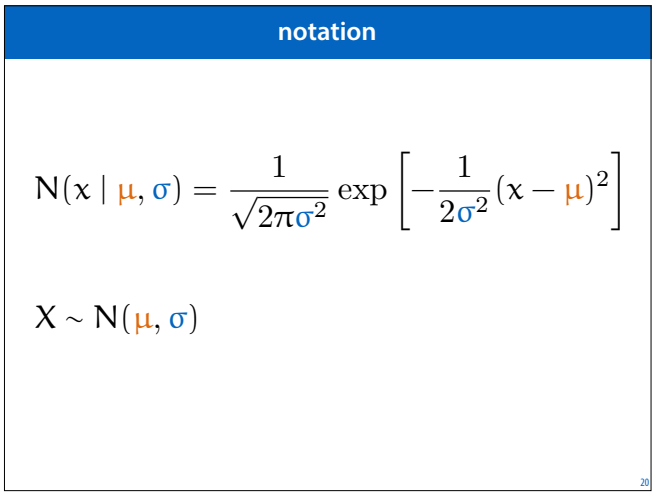


To control the mean, we introduce the parameter mu. Note that shifting a curve forward by mu points is the same as shifting the coordinates *backward* by three points . This is why we subtract mu from x.



Finally, to make this a probability density function, we need to make sure the area under the curve sums to one.

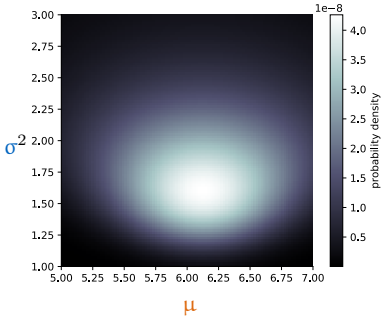
This is done by integrating over the whole real number line. If the result is Z, we divide the function at every point by Z. This gives us a function that sums to 1 over the whole of its domain.



A little notation: We use N(m,s) to refer to a specific probability distribution (for instance to say that X is distributed according to that distribution). We use N(x| m, s) to refer to the probability *density* function.

maximising the likelihood

X = [5, 4, 6, 7, 9, 8, 4, 6, 6]



21

Just like we did for the loss surface, we can plot the likelihood as a "surface" the model space. The brighter a point, the better the model fits the data (the higher the (log)probability)

maximum likelihood for the mean

$$\arg\max_\theta \ln p(X \mid \theta) = \arg\max_\theta \ln \prod_{x \in X} p(x \mid \theta)$$

$$= \arg\max_\theta \sum_x \ln p(x \mid \theta)$$

$$= \arg\max_{\mu,\sigma} \sum_x \ln \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}(x-\mu)^2\right]$$

$$= \arg\max_{\mu,\sigma} \sum_x \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(x-\mu)^2$$

22

If our data X consists of a series of **independent samples** from our assumed distribution, the total probability density is just the product of the individual probability densities (line 1). Working the product out of the logarithm, this becomes the sum of the individual log-densities (line 2).

We fill in the definition of the actual probability density function we're using (line 3). This function is the product of two factors (the division and the exponent) which become terms if we work them out of the logarithm. In the second term the exponent cancels against the logarithm.

We usually use a base-e logarithm, because it will cancel out against the base-e exponent in the probability density.

Earlier we used a base-2 logarithm, because we could then interpret the negative logarithm of the probability as a code length, but with continuous model spaces, we can't really do that any way.

maximum likelihood for the mean

$$\frac{\partial \ln p(X \mid \theta)}{\partial \mu} = \sum_x \frac{\partial\left[\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(x-\mu)^2\right]}{\partial \mu}$$

$$= -\frac{1}{2\sigma^2} \sum_x \frac{\partial (x-\mu)^2}{\partial \mu}$$

$$= -\frac{1}{\sigma^2} \sum_x (x-\mu)$$

23

We can now work out the derivative of the function we're trying to minimize, with respect to the parameter we're interested in.

maximum likelihood for the mean

$$-\frac{1}{\sigma^2}\sum_x (x-\mu) = 0$$
$$\sum_x (x-\mu) = 0$$
$$-\mu n + \sum_x x = 0$$
$$\mu = \frac{1}{n}\sum_x x$$

So far we've always been happy to just work out the derivative, and use gradient descent to find the optimum, but today, we'll pursue an analytical solution. We set the derivative to zero, and find that the maximum likelihood solution is the same as the old familiar mean.

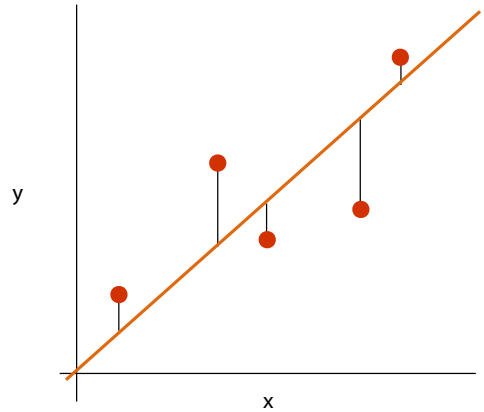
The maximum likelihood estimator of the variance works in the same way.

Note that this is essentially the same as the derivation of the mean we used in the fourth lecture: after we take the logarithm of the probability density everything disappears, except the squared errors, and the value that minimises the squared errors is the mean.

Historically, Gauss arrived at the normal distribution by following this derivation *backward*. He reasoned that (1) the mean of the data should be the natural estimator of some quantity V, given a series of measurements of that quantity. He then found that this was the most likely estimator given the data only if the *squares* of the differences (between V and the measurements) were minimised. From that he worked out that the probability density of the errors of the measurements should be proportional to the exponent of their negative square (i.e. the basic form of the normal distribution's density).

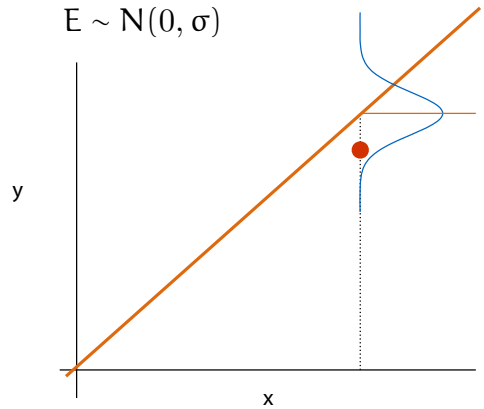
---

least squares regression



More surprisingly, we can also cast least-squares regression as a maximum likelihood problem.

$$Y = \mathbf{X}^T \mathbf{w} + b + E$$
$$E \sim N(0, \sigma)$$



We assume that our data was generated by a specific process. Somehow a random variable X was sampled, this sample was transformed by our linear model (**w**, b), and to the result of that, a scaler E of normally distributed random noise was added (zero mean, with some variance).

Note that we don't know what the distribution on X is. As it turns out, we don't need to know.

maximum likelihood for w and b

$$
\begin{aligned}
&\arg\max_{w,b} p(Y \mid X, w, b) \\
&= \arg\max_{w,b} \ln \prod_i N(y_i \mid x_i^T w + b, \sigma) \\
&= \arg\max_{w,b} \sum_i \ln \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}(x_i^T w + b - y_i)^2\right] \\
&= \arg\max_{w,b} -\sum_i \frac{1}{2\sigma^2}(x_i^T w + b - y_i)^2 \\
&= \arg\max_{w,b} -\frac{1}{2}\sum_i (x_i^T w + b - y_i)^2 = \arg\min_{w,b} \frac{1}{2}\sum_i (x_i^T w + b - y_i)^2
\end{aligned}
$$

27

As we can see here, all elements from the normal distribution disappear except the square difference between the predicted output and the actual output, and the objective reduces to least squares.

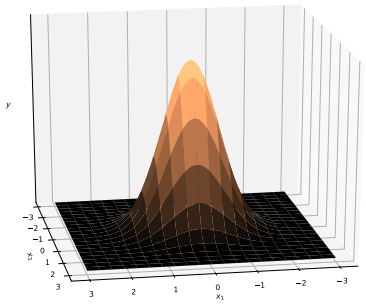
---

multivariate normal (MVN)

$$N(x \mid \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left[-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right]$$

28

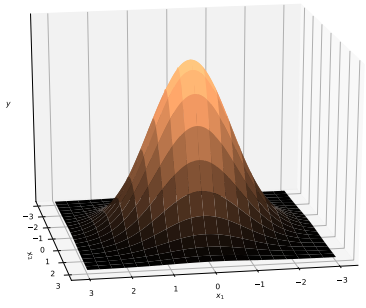
We can do the same thing in multiple dimensions. This gives us the multivariate normal distribution.



$$y = e^{-||x||^2}$$

29

We start by defining a curve that decays squared-exponentially in all directions. This creates a kind of "inflection circle" inside of which lie the expected outcomes.

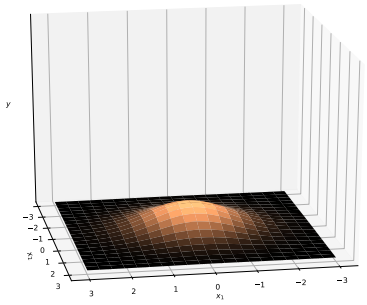


$$y = e^{-\frac{1}{2}x^T x}$$

30

To give the inflection circle radius 1, we rescale the exponent, as before.

Note that the square of the norm is equal to the dot product of a vector with itself.



$$y = \frac{1}{\sqrt{(2\pi)^d}} \exp\left[-\frac{1}{2}\mathbf{x}^T\mathbf{x}\right]$$

31

This time we'll normalize first, and then introduce the parameters.

This function is the probability density function of the *standard* MVN (zero mean, and variance one in every direction).

We can now define the probability density function of any other MVN by transforming it back to this one and rescaling the resulting density.

introducing parameters

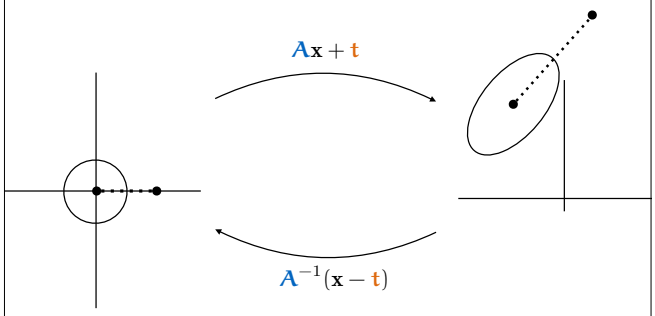
If

Y = **A**X + **t**, and

p(x) is the density of X, then

what is the density q(x) of Y?

32



$p(\mathbf{x})$ $\quad q(\mathbf{x}) = \frac{1}{|\mathbf{A}|} p(\mathbf{A}^{-1}(\mathbf{x} - \mathbf{t}))$

33

To introduce the parameters, we use a trick we've used before. If we can phrase our distribution q as an affine transformation **Ax** + **t** of another distribution p, and we have a density function for p, we can work out the density function q(x), by transforming **x** back, using the density under p, and correct for the amount that A inflates spaces.

The determinant expressed how much the matrix "inflates" space by transforming it: a (hyper)sphere of volume 1 is transformed into an (hyper)ellipse of volume |**A**| by **A**. Thus, it make sense that we need to scale by |**A**| to keep the area under the probability density function 1.

$$q(\mathbf{x}) = \frac{1}{|\mathbf{A}|} p(\mathbf{A}^{-1}(\mathbf{x} - \mathbf{t})) \qquad y = \frac{1}{\sqrt{(2\pi)^d}} \exp\left[-\frac{1}{2}\mathbf{x}^T\mathbf{x}\right]$$

$$y = \frac{1}{|\mathbf{A}|} \frac{1}{\sqrt{(2\pi)^d}} \exp\left[-\frac{1}{2}(\mathbf{A}^{-1}(\mathbf{x} - \mathbf{t}))^T(\mathbf{A}^{-1}(\mathbf{x} - \mathbf{t}))\right]$$

$$y = \frac{1}{\sqrt{|\mathbf{A}\mathbf{A}^T|}} \frac{1}{\sqrt{(2\pi)^d}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mathbf{t})^T\mathbf{A}^{-1^T}\mathbf{A}^{-1}(\mathbf{x} - \mathbf{t})\right]$$

$$y = \frac{1}{\sqrt{(2\pi)^d|\mathbf{A}\mathbf{A}^T|}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mathbf{t})^T(\mathbf{A}\mathbf{A}^T)^{-1}(\mathbf{x} - \mathbf{t})\right]$$

$$\Sigma = \mathbf{A}\mathbf{A}^T \quad \mu = \mathbf{t}$$

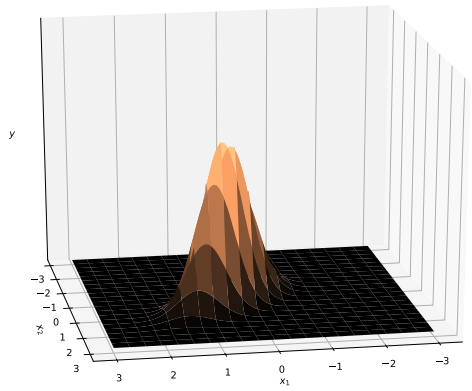
34

We fill in $\mathbf{A}^{-1}(\mathbf{x}\text{-}\mathbf{t})$ to transform our standard normal distribution pdf to the the pdf transformed by A and t. We set mu equal to t.

Using the basic properties of the determinant, the transpose and the inverse (you can look these up on wikipedia), we can rewrite the result to the pdf we expect.

$$N(x \mid \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)\right]$$

35



sampling

$N(0,1)$: see `numpy.random.randn`

$N(\mu, \sigma^2)$: $X\sigma + \mu$ with $X \sim N(0,1)$

$N^d(\mathbf{0},\mathbf{1})$: $\begin{pmatrix} X_1 \\ \vdots \\ X_d \end{pmatrix}$ with $X_i \sim N(0,1)$

$N^d(\mu, \Sigma)$: $AX + \mu$ with $X \sim N^d(\mathbf{0},\mathbf{1})$, $\Sigma = AA^T$

37

To sample from an MVN we can take the following approach.

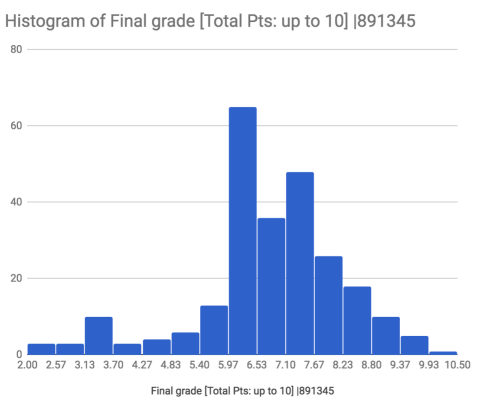
We'll take sampling form a univariate standard normal as read (it's usually done by an algorithm called the Box-Muller transform, if you're interested).

We can transform a sample from the standard normal distribution into a sample from a distribution with given mean and variance as shown above.

We can sample from the standard MVN by stacking d samples from the univariate normal in a vector.

We can then transform this to a sample from an MVN with any mean or covariance matrix by finding A and transforming as appropriate.

Gaussian mixture model



38

Here is the grade distribution from last year. It doesn't look very normally distributed (unless you squint), but we might be able to describe this distribution with a *mixture* of normal distributions.

GMM (with three components)

three components:

$$\mathrm{N}(\mu_1, \Sigma_1), \mathrm{N}(\mu_2, \Sigma_2), \mathrm{N}(\mu_3, \Sigma_3)$$

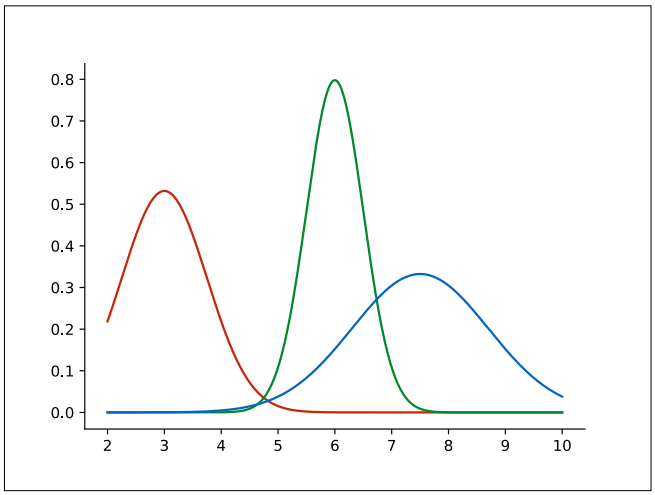
three weights

$$w_1, w_2, w_3 \quad \text{with} \quad \sum w_i = 1$$

39

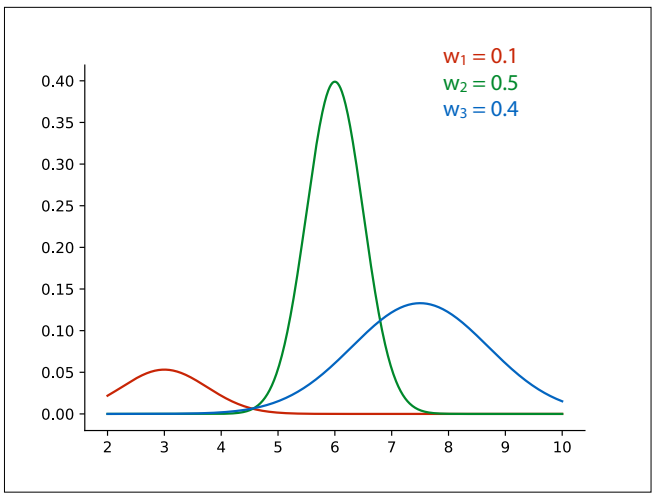
Here is how to define a mixture model.

If we were to sample a number from this model, we would

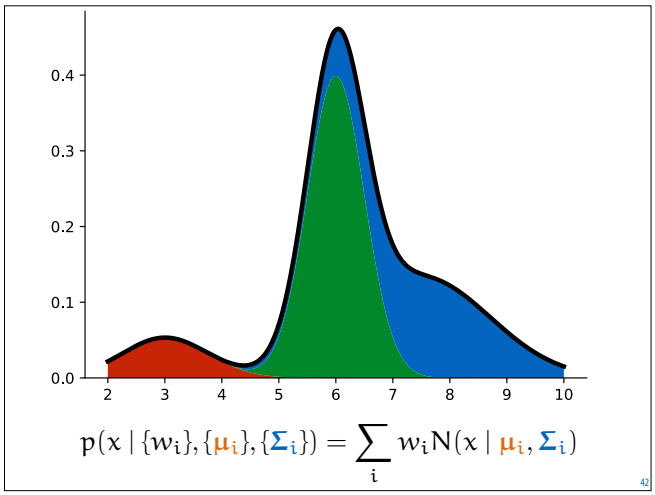


Let's take three component Gaussians.

We'll mostly look at the model in 1D, but it works the same for any dimensionality.



We scale each by a weight.



42

Then this is the probability density of the mixture model. The sum of the densities provided by the scaled components

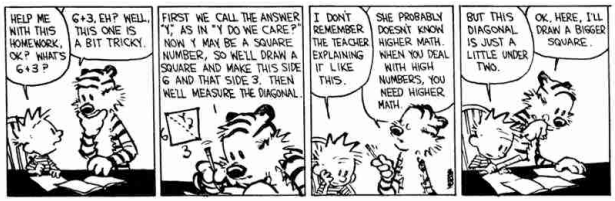
maximum likelihood

$$\arg\max_{\{w_i\},\{\mu_i\},\{\Sigma_i\}} \sum_x \ln \sum_i w_i N(\mathbf{x} \mid \mu_i, \Sigma_i)$$

Here we face a problem: there's a sum inside a logarithm. We can't work the sum out of the logari, which mean we won't get a nice formulation of the gradient. We can do it anyway, and solve by gradient descent, but we can't set the gradient equal to zero to get an analytical solution.

After the break we'll discuss the **EM algorithm,** which gives us an alternative way to fit a mixture of Gaussians.

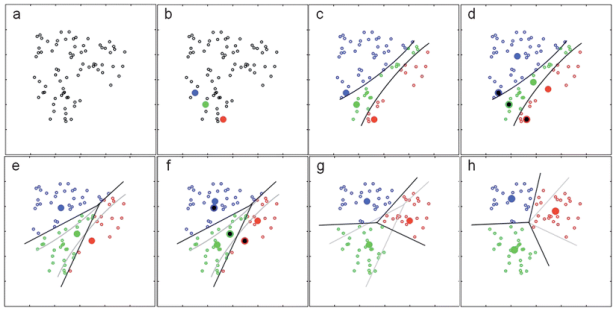
break



alternating optimization

Find an optimum by fixing one aspect of the problem, optimizing, fixing another, optimising and so on.

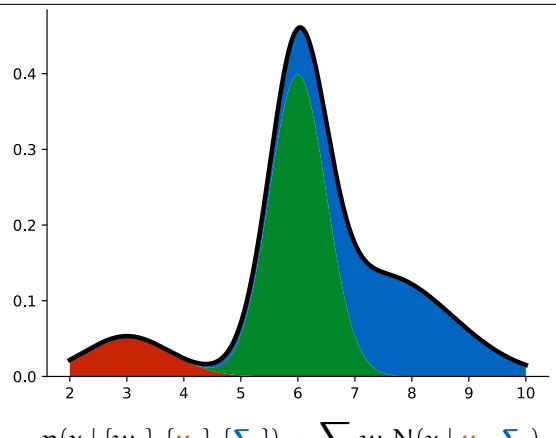
k-means



k-means is a clustering algorithm. It's probably the simplest example of an alternating optimization procedure.

The model works by picking k random *mean points*, that each represent a *cluster*. The data points are then assigned to the cluster who mean point is closest. We then throw away the original mean points, and compute new ones as the actual mean of each cluster. We then "re-color" the points and iterate until convergence.

In k-means clustering, each point belongs only one cluster. If we give the clusters a "soft reponsibility" for each point, and fit a multivariate normal distribution for each cluster, we get the EM algorithm.

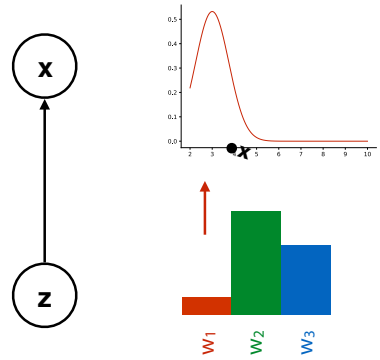


$$p(x \mid \{w_i\}, \{\mu_i\}, \{\Sigma_i\}) = \sum_i w_i N(x \mid \mu_i, \Sigma_i)$$

47

Back to the GMM. To fit this to data we can use what is probably the most famous alternating optimization method that exists: the **expectation maximisation** algorithm.

hidden variable model



48

The GMM is an example of a *hidden variable model*: the data is produced by sampling z, and then sampling x based on z, but we observe only x. z is called the hidden, or latent variable.

"completing" the data

If we knew **z** (we had the complete data), ML fitting would be easy.

Can we just marginalise **z** out?

$$p(X \mid \theta) = \sum_z p(X, Z = z \mid \theta)$$

For two components, and just 30 points, this sum has a billion terms!

49

The EM algorithm operates by guessing a *single* good value for z, based on the current parameters, then estimating new parameters and, guessing z again and so on.

problems

$$\arg\max_{\{w_i\},\{\mu_i\},\{\Sigma_i\}} \ln \sum_i w_i N(x \mid \mu_i, \Sigma_i)$$

no closed-form solution, gradient descent is possible

note the sum inside the logarithm

50

EM: key insight

We can't optimise for θ and **z** together, but:

- Given some θ, we can compute p(**z** | **x**)
- Given **z**, we can optimise θ

51

EM (intuitive)

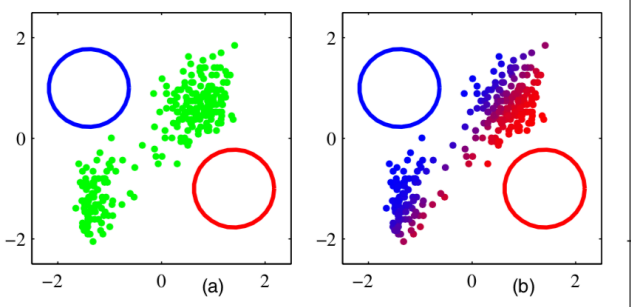
initialise components randomly

loop:

- expectation: assign soft *responsibilities* to each point
- maximization: fit the components to the data, weighted by responsibility.

52

The EM algorithm (for GMMs) expands on k-means by replacing the clusters with Gaussians, and by allowing points to "belong" to each Gaussian "to some degree". In other words, each Gaussian takes a certain *responsibility* for each point.

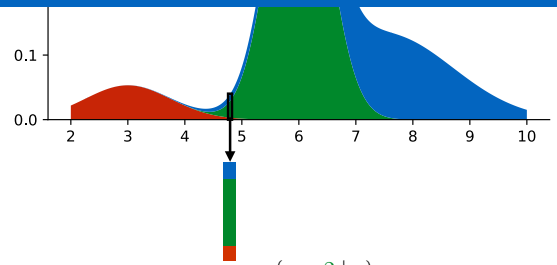


53

Here we see EM in action. We start with two random Gausssians and color the points by how much responsibility each component takes. The blue points are mostly claimed by the blue component, the red points are mostly claimed by the red, and the purple points in the middle have the responsibility divided equally between the components.

source: Machine Leaerning and Pattern Recognition, Christopher Bishop.

responsibilities (given the model parameters)



$$r_x^2 = \frac{p(z = 2 \mid x)}{p(z = 1 \mid x) + p(z = 2 \mid x) + p(z = 3 \mid x)}$$

$$r_x^i = \frac{p(z = i \mid x)}{\sum_j p(z = j \mid x)}$$

54

Here is how we define the responsibility.

In this case, the green component takes most of the responsibility for point x.

model parameters (given the responsibilities)

$$n_i = \sum_x r_x^i$$

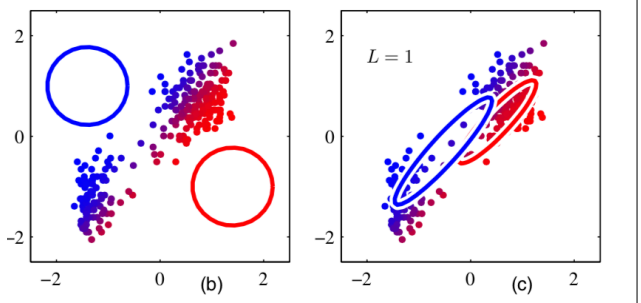
$$\mu_i = \frac{1}{n_i} \sum_x r_x^i x$$

$$\Sigma_i = \frac{1}{n_i} \sum_x r_x^i (x - \mu_i)(x - \mu_i)^T$$

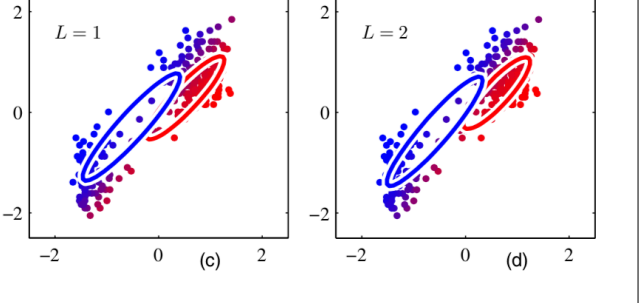
$$w_i = \frac{n_i}{n}$$

55

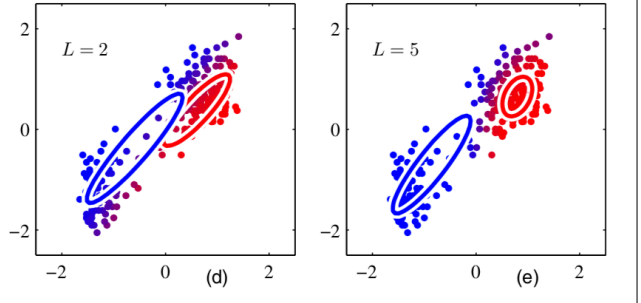
We compute a weighted mean and a weighted variance for each of the components, based on the responsibilities.



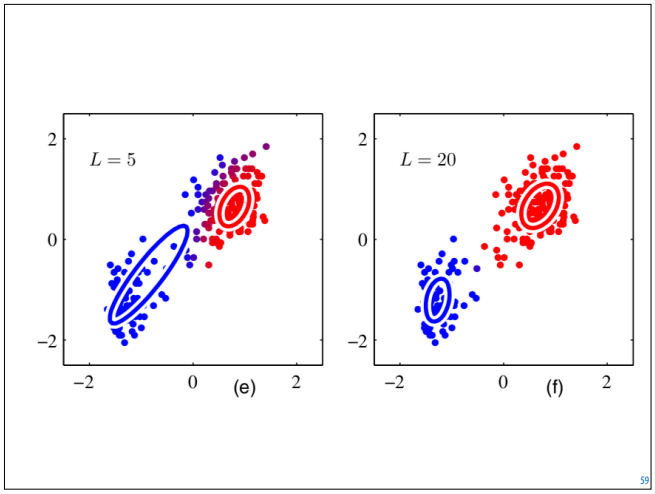
56



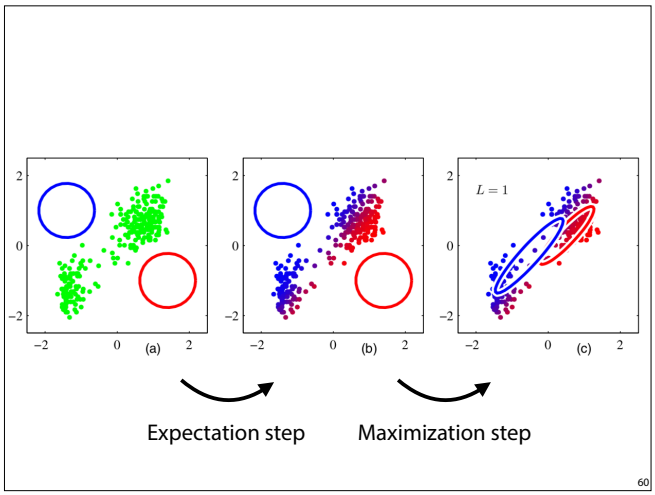
57



58



59



60

EM (formal treatment)

- Allows us to prove that EM converges (to a local optimum)
- Shows that the sample mean/variance weighted by the responsibilities are the correct solutions.
- Provides a decomposition that we can reuse for other hidden variable models.

61

a very useful decomposition

$$\ln p(\mathbf{x} \mid \theta) = L(q, \theta) + KL(q, p)$$

with :

$p = p(\mathbf{z} \mid \mathbf{x}, \theta)$

$q(\mathbf{z} \mid \mathbf{x})$ any approximation to $p(\mathbf{z} \mid \mathbf{x})$

$KL(q, p)$ Kullback-Leibler divergence

$$L(q, \theta) = \mathbb{E}_q \ln \frac{p(\mathbf{x}, \mathbf{z} \mid \theta)}{q(\mathbf{z})}$$

62

The likelihood p(x|theta), the one that we want to optimize, but that's too expensive to compute, can be rewritten for any approximation to p(z|x).

The term KL(q, p) indicates how good an approximation q is for p. The term L(q, theta) is whatever is left over.

This decomposition is very useful, and we'll see it again in the next lecture.

$$\ln p(\mathbf{x} \mid \theta) = L(q, \theta) + KL(q, p)$$

$$\begin{aligned}
&\mathbb{E}_q \ln \frac{p(\mathbf{x}, \mathbf{z} \mid \theta)}{q(\mathbf{z})} - \mathbb{E}_q \ln \frac{p(\mathbf{z} \mid \mathbf{x}, \theta)}{q(\mathbf{z})} \\
&= \mathbb{E}_q \ln p(\mathbf{x}, \mathbf{z} \mid \theta) - \mathbb{E}_q \ln p(\mathbf{z} \mid \mathbf{x}, \theta) \\
&= \mathbb{E}_q \ln \frac{p(\mathbf{x}, \mathbf{z} \mid \theta)}{p(\mathbf{z} \mid \mathbf{x}, \theta)} = \mathbb{E}_q \ln \frac{p(\mathbf{z} \mid \mathbf{x}, \theta) p(\mathbf{x} \mid \theta)}{p(\mathbf{z} \mid \mathbf{x}, \theta)} \\
&= \mathbb{E}_q \ln p(\mathbf{x} \mid \theta) = \ln p(\mathbf{x} \mid \theta)
\end{aligned}$$

63

Here is the proof that this decomposition holds. It's easiest to work backwards. We fill in our statement of the L and KL terms, and rewrite to show that they're equivalent to ln p(x|theta).

---

q =

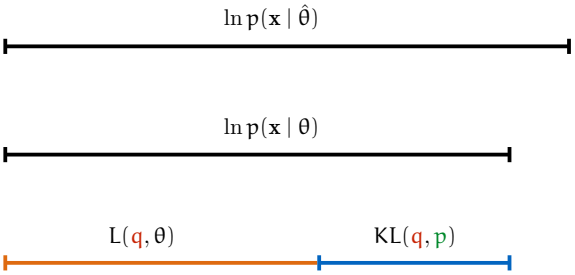
x1	0.1	0.4	0.5
x2	0.8	0.1	0.1
x3	0.3	0.6	0.1
x4	0.4	0.2	0.4

64

In our case the q function is an approximation q(z|x) to the true conditional probability p(z|x). We can't compute the true p(z|x), because we don't know the true model parameters. q(z|x) is the approximation using our current guess for the model parameters.

---

for *any* q

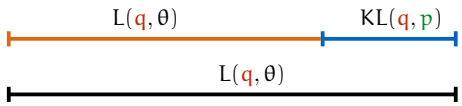


65

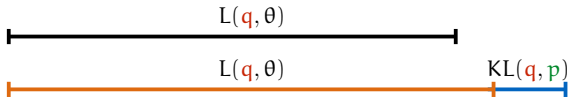
This is what this rewriting gives us. We have the probability of our data under the optimal model (top line) and the probability of our data under our current model (middle line). For any q, the latter is composed of two terms.

EM

E: Choose q so that KL(q, p) = 0. Keep θ fixed.



M: Choose θ to maximise L. keep q fixed.



66

The EM algorithm consists of two steps.

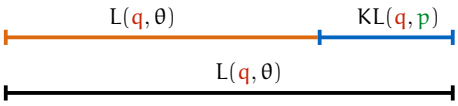
In the first we choose our q so that the KL divergence term is minimized.For the GMM algorithm we can compute p explicitly (these are the responsibilities), so we can set q equal to that and eliminate the KL divergence term entirely.

In the second step, we choose new parameters θ, to optimise the L-term. p has now changed, to the KL term is no longer zero.

Since both of these steps either keep p(**x**|θ) the same, or increase it, we have just proves that the EM algorithm converges (to a local minimum).

E step

E: Choose q so that KL(q, p) = 0. Keep θ fixed.



q(z|p) = p(z|x, θ)

The E step is easy to work out. The KL divergence is equal

M step

M: Choose θ to maximise L. keep q fixed.

$$\arg\max_{\theta} \sum_{z} q(z) \ln \frac{p(x, z \mid \theta)}{q(z)}$$

$$= \arg\max_{\theta} \sum_{z} q(z) \ln p(x \mid z, \theta) p(z \mid \theta) - \sum_{z} q(z) \ln q(z)$$

Note that the sum is now outside the logarithm. That means we can work out an optimal solution for the model parameters given the current q.

We won't show you the detailed rewriting, but if you take the function at the top, take its derivatives and set the equal to zero, they work out to the parameter values we expect for the EM algorithm

1D example

$$\arg\max_{\theta} \sum_{z} q(z) \ln p(x \mid z, \theta)\, p(z \mid \theta)$$

$$= \arg\max_{\{\mu_i, \sigma_i, w_i\}} \sum_{i,x} q(i \mid x) \ln N(x \mid \mu_i, \sigma_i) w_i$$

$$\arg\max_{\mu_1} \sum_{i,x} q(i \mid x) \ln N(x \mid \mu_i, \sigma_i) w_i$$

$$= \arg\max_{\mu_1} \sum_{i,x} r_x^i \ln N(x \mid \mu_i, \sigma_i) w_i$$

$$= \arg\max_{\mu_1} -\frac{1}{2} \sum_{i,x} r_x^i (x - \mu_i)^2 + \sum_{i,x} r_x^i \ln w_i$$

$$\arg\max_{\mu_1} -\frac{1}{2} \sum_{i,x} r_x^i (x - \mu_i)^2 \cdot$$

$$-\frac{1}{2} \sum_{i,x} r_x^i \frac{\partial (x - \mu_i)^2}{\partial \mu_1} = -\frac{1}{2} \sum_{x} r_x^1 \frac{\partial (x - \mu_1)^2}{\partial \mu_1}$$

$$= -\sum_{x} r_x^1 (x - \mu_1)$$

$$= -\sum_{x} r_x^1 x + \sum_{x} r_x^1 \mu_1 = -\sum_{x} r_x^1 x + \mu_1 \underbrace{\sum_{x} r_x^1}_{n_1}$$

$$-\sum_{x} r_x^1 x + \mu_1 n_1 = 0$$

$$\mu_1 = \sum_{x} r_x^1 / n_1$$

M step

M: Choose θ to maximise L. keep q fixed.

$$\arg\max_{\theta} \sum_{z} q(\mathbf{z}) \ln p(\mathbf{x} \mid \mathbf{z}, \theta) p(\mathbf{z} \mid \theta)$$

$$n_i = \sum_x r_x^i$$

$$\mu_i = \frac{1}{n_i} \sum_x r_x^i \mathbf{x}$$

$$\Sigma_i = \frac{1}{n_i} \sum_x r_x^i (\mathbf{x} - \mu_i)(\mathbf{x} - \mu_i)^T$$

$$w_i = \frac{n_i}{n}$$

Note that the sum is now outside the logarithm. That means we can work out an optimal solution for the model parameters given the current q.

We won't show you the detailed rewriting, but if you take the function at the top, take its derivatives and set the equal to zero, they work out to the parameter values we expect for the EM algorithm

What's the point?

clustering

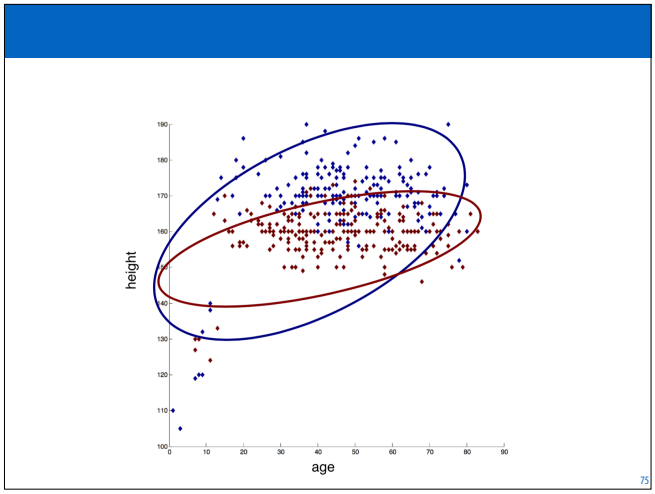
Fraud detection, targeting treatment, customer segmentation

- All unsupervised methods. Mostly useful for *exploratory* analysis.

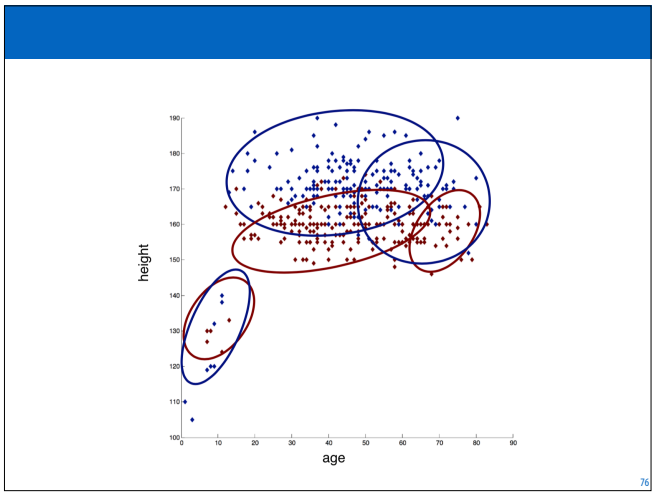
classification

Build a Bayes classifier. Fit an EM model to the points for each class, and classify by maximum

- This is a non-naive Bayes classifier. The features are not independent at all.



Here's what that looks like with a single Gaussian per class



With multiple Gaussians, we can fit the shape of the data more naturally.

summary

Normal distributions: very helpful building block for ore complex distribution

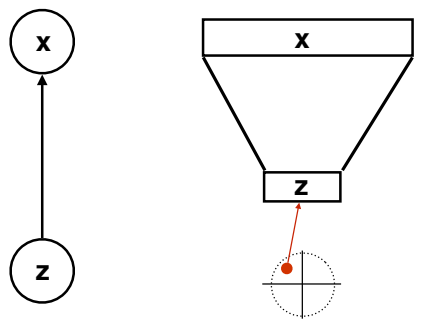
Maximum Likelihood: reasonable criterion for fitting models. Often corresponds to existing methods (sample mean, least squares regression)

Gaussian mixture models: Combine Gaussians to represent more complex shapes.

EM: Algorithm often associated with GMMs. Can be use to fit *any* hidden variable model.

77

Next lecture: hidden variable models with neural networks in the middle.



mlcourse@peterbloem.nl